# DiveScan: um módulo para Recuperação de dados por Similaridade com Diversidade em Postgres

**Anna Júlia Costa Lauton**[1], **Agma Juci Machado Traina**[1], **Caetano Traina Jr.**[1]

[1]Instituto de Ciências de Matemática e de Computação – Universidade de São Paulo (USP)
Avenida Trabalhador São-carlense, 400 - Centro – CEP: 13566-590 – São Carlos - SP

annalauton@usp.br, {agma, caetano}@icmc.usp.br

***Abstract.*** *Similarity queries with diversity are useful for applications that require results avoiding redundancy, such as medical diagnosis, recommendation systems, and image retrieval. However, the Database Management Systems still offer limited support for this type of query. This work presents the DiveScan module for Postgres, which executes such queries using the FM or BRID algorithms, extended to use GiST indexes to speed up the searches. Experimental evaluation shows that the indexed approach significantly outperforms sequential scanning in various scenarios, substantially reducing execution time.*

***Resumo.*** *Consultas por similaridade com diversidade são úteis em aplicações que demandam resultados que evitam redundâncias, como no diagnóstico médico, sistemas de recomendação e busca por imagens. Entretanto, os Sistemas de Gerenciamento de Bases de Dados (SGBDs) ainda oferecem suporte limitado a consultas desse tipo. Este trabalho apresenta o módulo DiveScan, uma extensão para o SGBD Postgres que executa tais consultas com os algoritmos FM ou BRID, e pode utilizar índices GiST para acelerar a busca. A avaliação experimental indica que a abordagem indexada supera a varredura sequencial em diferentes cenários, reduzindo significativamente o tempo de execução.*

## 1. Introdução

Suponha que um grande hospital possua um conjunto de prontuários que registra muitos anos de atendimentos aos seus pacientes, e que um profissional da saúde queira recuperar prontuários parecidos aos de um paciente sendo atendido, a fim de identificar os tratamentos que deram os melhores resultados. A busca precisa utilizar critérios de comparação por similaridade, pois dificilmente existem prontuários iguais. No entanto, usar apenas similaridade em conjuntos muito grandes recupera muitos prontuários bastante similares entre si, causando dificuldade para o profissional identificar as nuances que diferenciam as diversas condições clínicas envolvidas.

Essa situação, típica de sistemas de apoio ao diagnóstico médico, ocorre também em diversas outras áreas, como sistemas de recomendação, reconhecimento facial, detecção de plágio, pesquisas jurídicas e busca semântica em corpora. Conceitualmente, essa dificuldade pode ser reduzida associando critérios para induzir alguma diversidade nas respostas às consultas por similaridade, utilizando Consultas por Similaridade com Diversidade (*Diversified Similarity Queries* – DSQ). Sendo um problema de recuperação de dados, ele deve ser tratado no âmbito dos Sistemas de Gerenciamento de Bases de Dados (SGBDs). Portanto, algumas das questões a serem tratadas são: (a) como expressar

DSQs? (b) como representar os dados em um SGBD para consultas com DSQs? (c) como executar DSQs em um SGBD? e (d) como acelerar a execução de DSQs em um SGBD?

A recuperação de dados por similaridade é importante em muitas áreas de pesquisa, incluindo Recuperação da Informação [Hambarde and Proença 2023], Aprendizado de Máquina [Yang et al. 2024], Processamento de Linguagem Natural [Chandrasekaran and Mago 2021] e Análise de Dados [Shu and Ye 2023]. Existem muitas pesquisas focadas nos problemas específicos de cada área, e ferramentas também específicas e desconexas restritas às necessidades de uma ou poucas áreas e aplicações. Atualmente, existem poucos recursos para executar consultas por similaridade em SGBDs, e praticamente nenhum quando tais consultas também devem contemplar diversidade.

Neste estado da arte, este trabalho propõe um módulo para o SGBD relacional de código aberto Postgres, que provê mecanismos para recuperação de dados por similaridade. O módulo permite que analistas e desenvolvedores expressem DSQs com facilidade e flexibilidade sobre quaisquer dados armazenados passíveis de busca por similaridade, executando-as de maneira eficiente e integradas aos demais recursos do SGBD.

## 2. Conceitos e trabalhos relacionados

**Comparações por similaridade em SGBDs.** Em SGBDs, atributos escalares $\mathsf{A}$ (tais como números, textos ou datas) são usualmente comparados por operadores de comparação baseados em identidade ou ordem, denotados por $\theta \in \{=, \neq, <, >, \leq, \geq\}$. Atributos complexos $\mathsf{S}$ são usualmente comparados com operadores baseados em similaridade, mas para isso é necessário definir como calcular a similaridade, o que, em geral, é feito indicando uma função (ou coeficiente) de distância $f_d$ [Gupta et al. 2025]. As consultas por similaridade são executadas no espaço de distâncias $M = \langle \mathbb{S}, f_d \rangle$ onde $\mathbb{S}$ é o domínio do atributo $\mathsf{S}$ ($Dom(\mathsf{S}) = \mathbb{S}$) e $f_d$ é a função de distância associada. O domínio ativo de um atributo $\mathsf{S}$ em uma relação $\mathsf{T}$ é representado por $\mathbf{S} = \overset{*}{Dom}(\mathsf{S})$, onde $\mathbf{S} \subseteq \mathbb{S}$ é o subconjunto de valores $t[\mathsf{S}] \in \mathbb{S}$ que estão presentes em pelo menos uma tupla $t$ da relação $\mathsf{T}$. Em geral são definidos dois operadores de comparação por similaridade. Assim, dado um predicado $(s_1\ \theta_s\ s_2)$ sobre um par de elementos complexos $s_1, s_2 \in \mathbb{S}$, com $s_1$ correspondendo a um elemento de referência (também chamado centro da consulta):

(a) O operador de comparação pelos $k$-vizinhos mais próximos ($k$-$NN$: *$k$-nearest neighbors*) $\theta_s = k\text{-}NN\,[f_d, k]$ retorna VERDADE para a comparação $(s_1\ k\text{-}NN\,[f_d, k]\ s_2)$, $\forall s_i \in \mathbf{S}$, sempre que $f_d(s_1, s_2)$ for uma das $k$ menores distâncias $f_d(s_1, s_i)$;

(b) O operador de comparação por abrangência de similaridade ($Rng$: *similarity range*) $\theta_s = Rng\,[f_d, \xi]$ retorna VERDADE para a comparação $(s_1 Rng\,[f_d, \xi]\ s_2)$ sempre que $f_d(s_1, s_2) \leq \xi$, onde $\xi$ é um limite de similaridade.

**Diversificação de Resultados.** A resposta a uma consulta por similaridade pode incluir elementos muito semelhantes ao elemento central de busca, especialmente em grandes conjuntos de dados, que tendem a ter regiões do espaço com alta densidade, o que em geral ocorre justamente onde as consultas são mais frequentes. A falta de diversidade nas respostas pode ser um problema para o usuário, que precisa verificar uma grande quantidade de elementos e refinar frequentemente sua busca para encontrar informações relevantes. Portanto, é importante encontrar formas de retornar elementos similares ao elemento central da consulta, que também sejam diversos entre si. Na estratégia de diversificação

baseada em distância de separação [Skopal et al. 2009], o usuário especifica apenas a distância de separação mínima desejada entre os elementos da resposta. Por sua vez, na estratégia baseada em regiões de influência [Santos et al. 2013], a influência de um elemento sobre os demais é utilizada para estimar automática e dinamicamente a distância de separação mínima entre os elementos, sem precisar de conhecimento prévio do conjunto de dados nem do ajuste manual de parâmetros.

**Medidas de Influência.** Dados dois elementos distintos $s_i, s_j \in \mathbb{S}$, a influência mútua entre eles [Santos et al. 2013] é definida como $I(s_i, s_j) = 1/f_d(s_i, s_j)$. Assim, dado um elemento central de busca $s_q \in \mathbb{S}$, um vizinho diversificado $s_i \in \mathbf{S}$ e um objeto $s_j \in \mathbf{S}$ do conjunto de dados, quando $I(s_i, s_j) > I(s_q, s_j)$ então $s_j$ é mais influenciado por $s_i$ do que por $s_q$, e ele não deve estar no resultado, uma vez que $s_i$ já é um vizinho diversificado.

**Conjunto de Influência.** Dado um elemento central de consulta $s_q \in \mathbb{S}$, o conjunto de influência de um vizinho diversificado $s_i \in \mathbf{S}$ terá todos os elementos $s_j \in S \setminus \{s_i \cup s_q\}$ que estão mais distantes de $s_q$ do que $s_i$ e são mais influenciados por $s_i$ do que por $s_q$. Formalmente, esse conjunto é definido como:

$$\ddot{I}_{s_i,s_q} = \{s_j \mid s_j \in S \setminus \{s_i, s_q\}, \\ I(s_i, s_j) > I(s_i, s_q) \wedge I(s_i, s_j) > I(s_j, s_q) \wedge I(s_i, s_q) \neq I(s_j, s_q)\} \quad (1)$$

**Indexação em SGBDs.** Para acelerar a recuperação por determinadas condições de busca, é possível utilizar índices, como as árvores $B^+$-tree e R-tree. A estrutura Generalized Search Tree (GiST) [Hellerstein et al. 1995] permite implementar diversas árvores de busca como uma estrutura genérica, permitindo ao desenvolvedor tratar apenas dos detalhes específicos da sua estrutura, sem se preocupar com a implementação no banco de dados. A extensão nativa `CUBE`[1] do Postgres implementa a R-tree utilizando GiST, para representar e indexar pontos e hiper-cubos multidimensionais. A extensão disponibiliza as funções de distância: Euclidiana, Manhattan (ou City-Block) e Chebyshev.

**Trabalhos correlatos.** Várias propostas foram desenvolvidas para incorporar consultas por similaridade em SGBDs. O framework SimilarQL (SQL with Similarity) [Traina-Jr. et al. 2019] é construído sobre o Postgres visando facilitar a expressão dessas consultas em SQL definindo funções escritas em PL/pgSQL. O MIGUE-Sim (Multifunctional Integrated and User-friendly Engine for Similarity Queries) [Eleutério et al. 2024], além de criar funções para permitir executar consultas por similaridade no Postgres, utiliza o índice GiST R-tree para acelerar consultas aos $k$-vizinhos mais próximos. No entanto, nenhuma dessas soluções contempla a diversificação dos resultados.

Algoritmos relevantes no contexto da diversidade ainda não foram integrados aos SGBDs. O algoritmo *First-Match* ($FM$) [Skopal et al. 2009] visa recuperar $k$ elementos similares ao elemento central de consulta, mas distintos entre si, de acordo com uma distância de separação mínima ($\Phi$) dada pelo usuário. A técnica *Better Results with Influence Diversification* (*BRID*) [Santos et al. 2013] foi desenvolvida visando obter um conjunto de resultados diversificados atendendo aos requisitos dos conjuntos de influência. Essa abordagem possui duas variantes: o $BRID_k$, que retorna $k$ vizinhos diversificados e o $BRID_r$, que retorna elementos diversos separados pelo limiar de similaridade mínimo. Além disso, estudos exploram estratégias de indexação para acelerar consulta por simila-

[1] https://www.postgresql.org/docs/current/cube.html

ridade com diversidade, mas que são implementadas fora dos SGBDs. A técnica *diversity browsing* utiliza o algoritmo base *BRID* e combina o índice *VP-Tree* (*Vantage-Point Tree*) com a estratégia de busca *distance browsing* e critérios de consulta baseados em cobertura, realizando podas que evitam regiões redundantes [Jasbick et al. 2020]. Outra proposta incorpora diversidade na construção e busca usando o índice HNSW, levando em consideração regiões de baixa e alta dimensionalidade [Weber et al. 2024].

## 3. O Módulo DiveScan

Mesmo as poucas abordagens que integram consultas por similaridade ao núcleo dos SGBDs tratam apenas da similaridade, desconsiderando a diversidade dos resultados. Em consequência, as respostas tendem a incluir elementos com alta redundância entre si, dificultando a sua interpretação e análise. Visando oferecer suporte nativo a consultas por similaridade com diversidade em SGBDs de forma eficiente, propomos o módulo DiveScan[2] (explorar com `diversidade`).

DiveScan implementa a consulta de busca por similaridade diversificada $k$-$NDN$ ($k$-***n**earest **d**iversified **n**eighbors*) usando o comparador $\theta_s = k\text{-}NDN_{\mathsf{g}}[f_d, k]$. Ele retorna VERDADE para a busca $(s_1\ k\text{-}NDN_{\mathsf{g}}[f_d, k]\ s_q)$ sempre que $f_d(s_1, s_q)$ for uma das $k$ menores distâncias diversificadas por um algoritmo $\mathsf{g}$. Sua implementação usa somente recursos nativos do Postgres, com foco no uso do índice GiST R-tree para acelerar as buscas por similaridade sobre dados complexos, sem necessidade de módulos externos.

### 3.1. Estrutura de Dados e Indexação

Os objetos a serem comparados devem estar armazenados em uma tabela relacional, em atributos de um dos seguintes tipos principais: um vetor do tipo `CUBE`, um campo do tipo `POINT`, ou um vetor do tipo `DOUBLE PRECISION[]`. Essa representação permite flexibilidade na escolha da forma de acesso aos dados, seja por meio de varredura sequencial da relação ou por indexação, dependendo da função de distância e do algoritmo utilizado.

Para atributos do tipo `POINT`, é necessário aplicar a transformação `ll_to_earth`, que converte coordenadas geográficas (latitude e longitude) em coordenadas esféricas antes do cálculo da distância. Outras funções de distância podem ser definidas manualmente em PL/pgSQL para atributos $\mathsf{S}$ de tipos numéricos, para usar métricas personalizadas.

### 3.2. Algoritmos Implementados

O módulo DiveScan oferece suporte a três algoritmos de diversidade para executar consultas por similaridade com diversidade: $\mathsf{g} \in \{FM$ (*First-Match*)$, BRID_k$ e $BRID_r\}$, os quais permitem a implementação em PL/pgSQL para operar em espaços de distância e a utilização de indexação baseada em GiST R-tree. O pseudocódigo do algoritmo $\mathsf{g}$=*BRID*$_k$ implementado está apresentado no Algoritmo 1. Ele executa uma varredura ordenada pela distância ao centro da consulta $s_q$, etapa que pode ser acelerada por meio de índices GiST, e, para garantir diversidade, compara cada candidato aos elementos já selecionados no conjunto de resposta $\mathsf{T}_R$. A função de distância $f_d$ utilizada para ordenar e comparar os elementos depende do tipo do atributo vetorial $\mathsf{S}$ da relação. Quando $\mathsf{S}$ é do tipo `CUBE`, `DOUBLE PRECISION[]` ou `POINT`, podem ser utilizados os operadores nativos do Postgres: (a) `<->`: para distância Euclidiana ($L_2$), (b) `<#>`: para distância Manhattan ($L_1$) ou (c) `<=>`: para distância Chebyshev ($L_\infty$).

[2]Disponível em: https://github.com/AnnaLauton/DiveScan.git

**Algoritmo 1** Algoritmo $\mathtt{g} = BRID_k$ no Postgres

**Entrada:** Relação $\mathsf{T}$, atributo vetorial $\mathsf{S}$ (do tipo `CUBE`, `POINT` ou `DOUBLE PRECISION[]`) com as *features* dos elementos, elemento central $s_q$, $k$ e função de distância $f_d$
**Saída:** Tabela $\mathsf{T}_R$ com os vizinhos diversificados

1: **crie** tabela temporária $\mathsf{T}_R(\mathsf{S}_R)$
2: **para** cada tupla $t_i \in \mathsf{T}$, ordenada por $f_d(t_i[\mathsf{S}], s_q)$ **faça**
3: **se** $|\mathsf{T}_R| = k$ **então**
4: **break**
5: **fim se**
6: **defina** $t_i[\mathsf{S}]$ como `diverso`
7: **para** cada tupla $r_j \in \mathsf{T}_R$ **faça**
8: **se** $f_d(t_i[\mathsf{S}], r_j[\mathsf{S}_R]) \leq f_d(s_q, r_j[\mathsf{S}_R])$ e $f_d(t_i[\mathsf{S}], r_j[\mathsf{S}_R]) \leq f_d(s_q, t_i[\mathsf{S}])$ **então**
9: **defina** $t_i[\mathsf{S}]$ como: `não_diverso`
10: **break**
11: **fim se**
12: **fim para**
13: **se** $t_i[\mathsf{S}]$ é `diverso` **então**
14: **insira** $t_i[\mathsf{S}]$ em $\mathsf{T}_R(\mathsf{S}_R)$
15: **fim se**
16: **fim para**
17: **retorna** $\mathsf{T}_R$

Para os demais algoritmos disponíveis no módulo DiveScan, são aplicadas adaptações sobre o Algoritmo 1 básico. De maneira resumida, elas correspondem a:

(a) *BRID$_r$*: recebe o limiar $\epsilon$ como parâmetro, no lugar do parâmetro $k$ e substitui a linha 3 por: **se** $f_d(t_i[\mathsf{S}], s_q) > \epsilon$ **então**.
(b) *First-Match* ($FM$): além do valor do $k$, recebe como parâmetro a distância de separação $\Phi$ e modifica a linha 8 por: **se** $f_d(t_i[\mathsf{S}], r_j[\mathsf{S}_R]) < \Phi$ **então**.

Cada um dos três algoritmos foi implementado em PL/pgSQL como uma função de seleção $k$-$NDN$: `SELECT_KNNFM`, `SELECT_KNNBridR` e `SELECT_KNNBridk`. Por exemplo, uma consulta que busque os 10-vizinhos-mais-próximos-diversificados por *BRID$_k$*, de uma dada coordenada geográfica do atributo `Coord` no conjunto `Inside Airbnb` (vide descrição na Seção 4), usando a distância Euclidiana pode ser expressa como:

```
SELECT *
   FROM SELECT_KNNBridK('Airbnb', 'Coord',<centro> ,10, 'Euclidean');
```

## 4. Avaliação Experimental

Esta seção compara o custo computacional para executar o módulo DiveScan com os algoritmos *BRID$_k$*, *BRID$_r$* e *FM*, em modo sequencial e indexado. Foram usados três conjuntos de dados com diferentes características: `Inside Airbnb`[3] com coordenadas geográficas de $39{,}919$ anúncios da cidade do Rio de Janeiro em dezembro de 2024; `Glove`[4] com $400{,}000$ vetores de 50 dimensões, treinado com *tokens* da Wikipedia 2014 e Gigaword 5; e `Corel`[5] com $68{,}040$ vetores de 32 dimensões extraídas de imagens da coleção Corel; A Figura 1 mostra o tempo médio de *clock* (miliseg., escala log) necessário para a

[3]Inside Airbnb: https://insideairbnb.com/get-the-data/
[4]Global Vectors for Word Representation: https://nlp.stanford.edu/projects/glove/
[5]Corel Image Features: https://archive.ics.uci.edu/dataset/119/corel+image+features

execução dos algoritmos usando valores de $k \in \{5, 10, 50, 100\}$. Cada consulta foi executada 20 vezes, centrada em elementos aleatórios, sempre fazendo um *warm-up* prévio para atenuar os efeitos do cache do SGBD, para maior confiabilidade das medições.

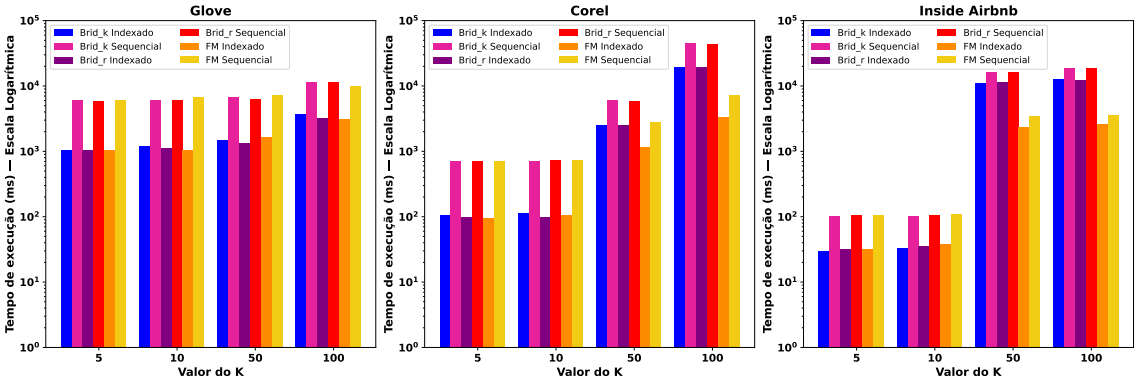


**Figura 1. Tempo médio de execução (em escala logarítmica) dos algoritmos**

Os resultados indicam que a abordagem indexada ganha em desempenho em todos os cenários analisados. Observa-se que o dataset `Glove`, de maior cardinalidade e dimensionalidade, apresenta redução média de 78%, com valores entre 67% a 85%. Esse resultado evidencia o impacto positivo dos índices conforme o volume de dados cresce, reduzindo o custo da busca sequencial. O dataset `Corel` também apresenta ganhos significativos, com redução média de 71%, variando de 54% a 87%. Igualmente, o conjunto `Inside Airbnb`, que tem a menor cardinalidade e dimensionalidade, tem redução média de 50%, variando de 28% a 71%, indicando que os índices contribuem também aqui.

Os resultados mostram que o método indexado supera o método sequencial por margem significativa, reduzindo o tempo entre 28% a 87%. Ressalta-se que o algoritmo *FM* geralmente apresenta tempos de execução menores, mas ele utiliza uma distância de separação pré-definida, prejudicando os resultados.

## 5. Conclusões

Este trabalho apresenta o módulo DiveScan, uma solução nativa para Postgres voltada principalmente a facilitar a execução de consultas por similaridade com diversidade, o que previamente não é atendido por nenhum SGBD. A abordagem proposta se destaca por não depender de recursos externos à infraestrutura do SGBD, permitindo a integração direta dessa nova classe de consultas com todas as demais já disponíveis nas operações de comparação em SQL. Além disso, os resultados experimentais mostram também que índices GiST, já disponíveis no gerenciador, podem ser usados para melhorar significativamente o tempo de execução das consultas, principalmente em dados de alta cardinalidade e dimensionalidade. Como trabalho futuro, devem ser exploradas outras estruturas de indexação, assim como novas variantes de consultas por similaridade com diversidade e métodos de diversificação de resultados.

## Agradecimentos

Este trabalho foi apoiado pela Fundação de Amparo à Pesquisa do Estado de São Paulo (FAPESP) (Processos 23/18026-8 e 24/13328-9), Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES).

## Referências

Chandrasekaran, D. and Mago, V. (2021). Evolution of semantic similarity—a survey. *ACM Comput. Surv.*, 54(2):Article 41.

Eleutério, I. A. R., Cazzolato, M. T., Teixeira, L. R., Gutierrez, M. A., Traina, A. J. M., and Traina-Jr., C. (2024). Migue-sim: Speeding up similarity queries with native rdbms resources. In *Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing*, pages 321–328, New York, NY, USA. Association for Computing Machinery.

Gupta, S., Thakar, U., and Tokekar, S. (2025). A comprehensive survey on techniques for numerical similarity measurement. *Expert Systems with Applications*, 277:127235.

Hambarde, K. A. and Proença, H. (2023). Information retrieval: Recent advances and beyond. *IEEE Access*, 11:76581–76604.

Hellerstein, J. M., Naughton, J. F., and Pfeffer, A. (1995). Generalized search trees for database systems. In Dayal, U., Gray, P. M. D., and Nishio, S., editors, *International Conference on Very Large Databases (VLDB)*, pages 562–573, Zurich, Switzerland. Morgan Kaufmann.

Jasbick, D., Santos, L., de Oliveira, D., and Bedo, M. (2020). Some branches may bear rotten fruits: Diversity browsing vp-trees. In *Similarity Search and Applications*, pages 140–154. Springer.

Santos, L. F. D., Oliveira, W. D. d., Ferreira, M. R. P., Traina, A. J. M., and Traina Jr, C. (2013). Parameter-free and domain-independent similarity search with diversity. In Szalay, A., Budavari, T., Balazinska, M., Meliou, A., and Sacan, A., editors, *25th International Conference on Scientific and Statistical Database Management - SSDBM'2013*, pages 5–16, Baltimore, MD, USA. ACM.

Shu, X. and Ye, Y. (2023). Knowledge discovery: Methods from data mining and machine learning. *Social Science Research*, 110:102817.

Skopal, T., Dohnal, V., Batko, M., and Zezula, P. (2009). Distinct nearest neighbors queries for similarity search in very large multimedia databases. In Chan, C. Y. and Mitra, P., editors, *11th ACM International Workshop on Web Information and Data Management WIDM 2009*, pages 11–14, Hong Kong, China. ACM.

Traina-Jr., C., Moriyama, A., Rocha, G., Cordeiro, R., Ciferri, C. D. A., and Traina, A. (2019). The similarql framework: Similarity queries in plain SQL. In *Proceedings of the ACM Symposium on Applied Computing*, Limassol, Cyprus. ACM.

Weber, M., Silva-Leite, J., Santos, L., de Oliveira, D., and Bedo, M. (2024). Adicionando suporte à diversificação de resultados em índices hnsw considerando espaços de baixa e alta dimensionalidade. In *Anais do XXXIX Simpósio Brasileiro de Bancos de Dados*, pages 14–26. SBC.

Yang, P., Wang, H., Yang, J., Qian, Z., Zhang, Y., and Lin, X. (2024). Deep learning approaches for similarity computation: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 36(12):7893–7912.